Gremlin Design Decisions

Table of Contents

[Hardware 2](#_Toc309029133)

[Reasoning behind some core decisions 2](#_Toc309029134)

[Resolution 2](#_Toc309029135)

[Bit-depth 2](#_Toc309029136)

[BG’s 2](#_Toc309029137)

[Sprites 2](#_Toc309029138)

[Palettes 2](#_Toc309029139)

# Hardware

## Reasoning behind some core decisions

Some decisions had to be made with regards to the overall feature-set of the VGA engine. Here are some of the thoughts behind the decisions:

1. First of all, this had to be implementable on a Xilinx Spartan 3 FPGA, so size does matter. If a feature caused too much of the device to be used, I opted not to offer it.
2. If a feature required a clock-rate that was too high, I would have trouble meeting timing constraints, and the feature would be difficult or impossible to implement. Or the feature would work, but adding glue logic with the rest of the system would push the timing constraints over the limit. I opted for easy timing.

### Resolution

If the resolution was any higher, I ran into the following obstacles:

1. Each BG would have required twice as much VRAM for their maps (meaning that less VRAM would be available for other things). A 64x32 BG can only offer 512 x 256 pixels, so going with a higher resolution of 400x300 would have required a 64x64 map ☹ I wanted more BG’s, not bigger ones.

### Bit-depth

1. Offering multiple bit-depths caused my prototype to increase in size dramatically, so the final version offers only 4BPP, although I did offer a lot of palettes in exchange for this limitation.

### BG’s

1. I wanted at-least 3 BG’s. One foreground, one background, and a text layer on top of everything. The sprites lie between the foreground and background layers.
2. I was not interested in a massive ‘native’ map. In any reasonably complex game, you need to deal with off-screen map DMA’s anyway, in order to produce a massive scrolling area, so why not just use that strategy for all maps? VRAM is limited, so let’s keep it on-screen. A 64x32 cell map (512x256 pixels) is the smallest size that encapsulates the entire screen and gives enough vertical space to hide a sprite off-screen as well.

### Sprites

1. I wanted rotation and flipping. Therefore, the sprite should be square, for easiest handling.
2. There’s only so much VRAM for sprites, so large sprites would waste VRAM. I opted for many smaller sprites that you can stitch together with code.
3. There is only 16 scanlines offscreen (vertically) where you can hide a sprite without a rendering penalty, so a sprite had to be 16 pixels or shorter. Therefore, I opted for a 16x16 pixel sprite.

### Palettes

1. The smallest BRAM available in 12bit is 1K large in 12bit entries, so I looked for ways to effectively make use of the palette address space.
   1. I gave BG’s a limited per-cell palette option, as well as a flexible global palette option.
   2. I gave sprites a flexible per-sprite palette option.
2. In order to limit the size of the generated circuitry, I made the scanline buffer record a color-index and palette bank (totaling 4+6=10 bits) rather than a full 12bit RGB color. It also places the palette lookup circuitry on the VGA pixel plotting side as opposed to the rendering side, simplifying things on the rendering side as well. This small optimization may be unnecessary; I didn’t try the other way.